Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Małgorzata Sawitus
Nr albumu: 201116

Bazy danych astronomicznych

Praca magisterska
na kierunku INFORMATYKA
w zakresie BAZ DANYCH

Praca wykonana pod kierunkiem
dr hab. Jerzego Tyszkiewicza
Instytut Informatyki

czerwiec 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W pracy przedstawiono projekt aplikacji ułatwiającej dostęp do danych przechowywanych w relacyjnych bazach danych. Pomysł opiera się na tworzeniu formularzy opisujących zapytania w plikach XML'owych. Dzięki temu użytkownik końcowy nie musi znać języka SQL, by pobierać dane z bazy. Zmiana istniejących formularzy i dodanie nowych jest prostą czynnością. Do pracy dołączona jest implementacja aplikacji opartej na tym pomyśle. Duży nacisk położony został na to, aby była ona rozszerzalna i łatwo konfigurowalna, tak by mogła pracować z wieloma bazami danych. Wymagania dla aplikacji zostały zebrane na podstawie rozmów z członkami astrofizycznego projektu "Pi of the sky", przyszłymi użytkownikami.

Słowa kluczowe

baza danych, SQL, tworzenie zapytań, XML, dane astronomiczne

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. Information Systems
H.3 Information Storage and Retrieval
H.3.3 Information Search and Retrieval

Tytuł pracy w języku angielskim:

Astronomical databases

Spis treści

Spis rysunków

Rozdział 1

Wprowadzenie

Obecnie niemal każdy większy system informatyczny korzysta z bazy danych. Są one powszechnie stosowane w bardzo różnych dziedzinach nauki czy biznesu. Jednak na sposobie zbierania i przechowywania danych problemy się nie kończą. Bez wygodnych rozwiązań ich pobierania, przeglądania i analizy same dane na niewiele się zdadzą. Zdecydowanie najpowszechniejszym rodzajem bazy danych jest relacyjna baza danych. Z tym wiąże się popularność języka SQL, opracowanego specjalnie dla nich. Chcąc wydobyć dane z bazy, każdy stosuje bezpośrednio lub pośrednio (czasem układa je za nas aplikacja) zapytania SQL.

Dane zazwyczaj analizuje nie programista, lecz osoba znająca się na dziedzinie z jakiej są dane w bazie (np. astronomowie, fizycy, biolodzy). Zazwyczaj nie zna się ona na bazach danych, nie zna także języka SQL. Osoby takie potrzebują aplikacji, która ułatwi im dostęp do danych tj. wykonanie zapytania do bazy danych, dostosowaną do potrzeb wizualizację wyników zapytania, a także możliwość ich zapisu w stosownym formacie. Pomimo ogromnej popularności relacyjnych baz danych, wciąż brakuje na rynku prostych w obsłudze i praktycznych narzędzi wspomagających dostęp do nich. W przypadku wielu baz danych bywa, że ogromne ilości danych zajmują wyłącznie miejsce, podczas gdy przy wykorzystaniu odpowiednich narzędzi mogłyby się przyczynić do ciekawych odkryć.

Wygodny i, co najważniejsze, praktyczny interfejs byłby wielką korzyścią także dla tych użytkowników, którzy dobrze znają język SQL.

Tworzenie specjalnego interfejsu dla obsługi konkretnych danych nie zawsze jest konieczne. Często dużo lepszym, tańszym rozwiązaniem jest skorzystanie z gotowych aplikacji stworzonych do tego celu. Wówczas dodatkową zaletą jest krótszy czas wdrażania. Narzędzia tego typu muszą się cechować dużą elastycznością. Dobrze, gdy są łatwo rozszerzalne, by można je było dopasować do danego zastosowania.

W pracy zaprezentuję rozwiązanie, które wychodzi naprzeciw problemom opisanym powyżej. Opiszę system ułatwiający komunikację z bazą danych i wyszukiwanie w niej potrzebnych danych. Jego podstawowymi celami są:

- wygodny sposób wykonywania zapytań,
- odpowiedni do rodzaju wyniku sposób jego wizualizacji,
- zapis wyników w stosownym formacie.

Ponadto jego ważną cechą jest niezależność od zastosowania, łatwość dostosowania nie tylko do różnych baz danych, ale ogólnie w różnych projektach.

Podejście do problemu, które zaprezentuję, daje możliwości rozszerzenia systemu o wiele ciekawych, także nietypowych opcji.

Ważnym elementem pracy jest konkretna implementacja realizująca opisane rozwiązanie. Jej istotną cechą pod względem programistycznym jest obiektowe podejście i projekt umożliwiający łatwe dodawanie nowych właściwości.

Program został zaprojektowany w oparciu o potrzeby astrofizycznego projektu „Pi of the sky", w którym to jest obecnie wdrażany, co dodatkowo potwierdza jego praktyczność.

Na początku pracy zdefiniuję kluczowe pojęcia oraz wyjaśnię znaczenie używanych przeze mnie terminów. Następnie określę zadanie, które postawiłam sobie za cel na początku i które zrealizowałam w omawianym rozwiązaniu. Było ono kluczowe przy projektowaniu systemu. Potem zaprezentuję istniejące rozwiązania, opiszę ich wady i zalety. Warto zauważyć, że istnieje wiele rozwiązań na rynku, które wychodzą naprzeciw podobnym problemom, które różnicują się pod względem poziomu niezależności od zastosowania, trudności w obsłudze i zaawansowania analiz. Wreszcie opiszę dokładnie mój pomysł, zaczynając od najważniejszych cech i kończąc na dokładnej analizie implementacji. Następnie wymienię pomysły na dalszy rozwój mojego systemu t.j. opcje, które wychodzą poza zakres kluczowych, lecz w wielu przypadkach mogą się okazać praktyczne. Omówię także zastosowanie stworzonej przeze mnie aplikacji w projekcie astrofizycznym i przybliżę problemy, które dzięki niemu udało się rozwiązać. Na koniec zrobię krótkie podsumowanie. W dodatkach zawarłam listę dołączonych do pracy załączników, które pomogą w głębszym zrozumieniu działania programu. Do pracy dołączony jest także wcześniej wspomniany program oraz inne przydatne dla użytkownika dokumenty z nim związane.

Rozdział 2

Podstawowe pojęcia i założenia

2.1. Pojęcia

SQL (Structured Query Language) - Język zapytań dla relacyjnych baz danych.

XML (Extensible Markup Language) - Metajęzyk, pozwalający opisać znacznikami tekst w dokumentach tekstowych. Znaczenie elementów określa aplikacja. Poprawność dokumentu jest ściśle określona przez specyfikację.

XML Schema - Standard służący do definiowania języka (struktury dokumentu XML).

XML DOM - Standard dostępu i przetwarzania dokumentów XML. Reprezentuje dokument przez strukturę drzewiastą.

plugin (wtyczka) - Dodatek do programu, poszerzający jego możliwości, często pisany przez programistów nie tworzących aplikacji macierzystej.

2.2. Założenia

aplikacja - Logika razem z (przykładowym) interfejsem z niej korzystającym, cel tejże pracy.

wdrażanie aplikacji - Przystosowywanie aplikacji do użytkowania w konkretnym celu. Głównym zadaniem podczas wdrażania jest stworzenie plików określających parametry połączenia oraz pliki opisujące formularze.

użytkownik - Końcowy użytkownik aplikacji.

administrator systemu - Osoba, która zna język SQL i XML, potrafiąca napisać pliki konfiguracyjne o podanym schemacie. Administratorem może być osoba wdrażająca system.

formularz - Zbiór danych zawierający zapytanie SQL wraz z polami określającymi parametry zapytania. Formularze aplikacji są opisane w plikach XML.

wypełnienie formularza - Przykładowe dane odpowiadające parametrom zapytania. Takimi danymi mogą być wypełnione pola formularza, muszą być zgodne z nimi co do typu.

Rozdział 3

Postawienie zadania

W tejże pracy omówię narzędzie do pobierania danych z bazy. Cechy, które ma spełniać, wynikają z praktycznych problemów. Postaram się je teraz przybliżyć.

Jak wspomniałam we wstępie, system ma przede wszystkim pomagać w pracy z danymi przechowywanymi w relacyjnej bazie danych osobom nie znającym języka SQL, z czego wynika najważniejsze zadanie aplikacji: ma umożliwić definiowanie zapytań wykonywanych na bazie danych. W większości przypadków będą to zapytania o bardzo podobnej strukturze różniące się parametrami.

Poza liczną grupą osób, które nie znają SQLa istnieje spora grupa tych, którzy znają go pobieżnie. Dla nich budowanie prostych zapytań nie stanowi problemu, lecz pojawia się on, gdy zachodzi potrzeba tworzenia tych bardziej skomplikowanych. Często trudnością nie do przezwyciężenia staje się ułożenie optymalnych zapytań. Jednakże zrozumienie przez nich podstaw baz danych popycha w kierunku własnych poszukiwań w układaniu dodatkowych zapytań, nie ograniczają się tylko do tych standardowych.

W licznych projektach naukowych dane (a przynajmniej ich część) udostępnia się światu, czyli każdemu, kto byłby zainteresowany analizą danych z pewnej dziedziny. Osoby nie związane z danym projektem naukowym, często amatorzy w danej dziedzinie mogą początkowo nie zdawać sobie sprawy, jakie zapytania mogą dawać ciekawe rezultaty. Przydatnym dla nich są zdefiniowane wzorce zapytań, pozwalające pobrać dane z bazy danych, które zainteresują nawet „początkujących naukowców".

Po pobraniu danych aplikacja musi udostępnić wygodny sposób ich przeglądania. W zależności od rodzaju danych i upodobań użytkownika przydatne będzie wyświetlanie danych w postaci tekstowej (np. tabela) lub graficznej (różne rodzaje wykresów).

W wielu przypadkach osoby analizujące dane posługują się specjalistycznymi programami do ich przetwarzania. Dla użytkowników korzystających z dodatkowych narzędzi analitycznych niezbędna jest opcja zapisu wyniku wyszukiwania (często w określonym formacie).

Ze względu na nieograniczanie sposobu wykorzystania programu do konkretnych zastosowań musi on brać pod uwagę różne bazy danych, różne schematy baz, różne rodzaje danych. Zatem kolejnym wymaganiem będzie łatwa konfiguracja systemu, dostosowanie do różnych projektów. To wymaganie jest także stawiane przez projekty, które są w fazie zmian technologii (np. zmiana bazy danych).

Aplikacja może służyć do pracy z bardzo dużymi bazami danych, co powinno się wziąć pod uwagę przy jej projektowaniu. Przydatnym może się więc okazać wprowadzenie ograniczeń np. na maksymalną liczbę wyszukiwanych rekordów z bazy danych.

Jednym z podstawowych wymagań, często decydującym o korzystaniu z aplikacji jest wygoda posługiwania się nią. Dodatkowe opcje mogą znacznie usprawnić pracę z programem. W przypadku aplikacji umożliwiającej dostęp do danych będzie to:

- Możliwość zapisu wykonywanych operacji. Opcja ta może się przydać np. jeśli chcemy porównywać wyniki w czasowo zmieniających się lub przyrastających danych. Może też być ułatwieniem przy dzieleniu się z innymi osobami w projekcie ciekawymi wynikami (zamiast przesyłania dużej ilości danych można przesłać zapytanie je wyszukujące).
- Wykonywanie wielu zapytań jednocześnie. Opcja ta przydatna zwłaszcza, gdy konieczne jest dłuższe niż kilka sekund oczekiwanie na wynik z bazy.
- Łatwe przełączanie baz danych, na których użytkownik chce wykonywać zapytania. W przypadku, gdy istnieje kilka baz o tej samej strukturze można w łatwy sposób porównywać dane, które przechowują. Opcja ta przydaje się też w bardziej ogólnej sytuacji, gdy chcemy wykonywać różne zapytania w różnych bazach i przy analizie jednych danych korzystać z wyników z drugiej bazy.

Rozdział 4

Istniejące rozwiązania

W rozdziale tym przedstawię z perspektywy użytkownika kilka istniejących na rynku rozwiązań – aplikacji ułatwiających dostęp do danych w bazie. Wszystkie istniejące aplikacje znacząco różnią się od pomysłu zaprezentowanego w pracy, stąd też trudne jest ich porównanie z moją aplikacją. Powodem takiego stanu rzeczy jest brak narzędzi nastawionych na zwykłych użytkowników baz danych – tj. takich, którzy zainteresowani są przede wszystkim wykonywaniem zapytań do bazy danych.

Istniejące na rynku aplikacje opierają się na pomyśle, by użytkownik na początku pracy z programem podał parametry połączenia, po czym aplikacja wczytuje struktury istniejące w bazie danych. Teraz użytkownik może układać zapytania wybierając odpowiednie tabele i określając powiązania między nimi, tworząc poprzez wybieranie z listy możliwych operacji wymagane warunki zapytania.

4.1. DBxtra

Przykładem aplikacji zbudowanej w oparciu o taki pomysł jest DBxtra (zob. [DBX]). Jest to narzędzie służące przede wszystkim do tworzenia raportów. Jego głównym założeniem jest to, aby użytkownik nie musiał znać SQLa, z kolei użytkownikom znającym ma pomóc w układaniu trudniejszych zapytań. Możliwe jest tutaj łączenie się lokalnie lub zdalnie, aplikacja posiada wsparcie dla wielu rodzajów baz danych. Projektowanie zapytań odbywa się na zasadzie „drag & drop", tworzeniu zależności, wybieraniu pól wyjściowych, nadawaniu im przyjaznych nazw, dodawaniu wyrażeń i definiowaniu właściwości zapytania. Można stosować wyrażenia grupowania i agregacji tj: sum, count itd., jak również definiować dowolną liczbę warunków oraz określać porządki sortowania. Dodatkową opcją jest możliwość edytowania zapytania w postaci wygenerowanego zapytania SQL. Wynik zapytania można obejrzeć w prostym edytorze. Aplikacja posiada także udogodnienia w postaci drukowania wybranych kolumn, wyszukiwania rekordów i eksportowania do kilku formatów plików. DBxtra ma także inne własności (związane z raportowaniem), jednak są mniej znaczące dla porównania z pomysłem zawartym w mojej pracy.

4.2. Inne aplikacje

Istnieje wiele innych aplikacji, które działają na podobnej zasadzie, co powyżej opisana DBxtra. Są to m.in. Data Analyzer (zob. [MIC]), Visual SQL-Designer (zob. [VIS]), DBACentral (zob. [DBA]).

Niestety narzędzia te są dość skomplikowane i osobie słabo obeznanej z komputerami dużo czasu zajmuje ich opanowanie. Konieczna jest też w przypadku tychże aplikacji znajomość podstaw baz danych (relacje występujące pomiędzy tabelami), przez co w wielu przypadkach nie nadają się one do wykorzystania przez użytkowników, którzy nie mają pojęcia o faktycznym działaniu bazy danych. W przypadku opisanych aplikacji użytkownik końcowy zmuszony jest do zapoznania się ze strukturą bazy danych (przykładowo: nie ma tu łatwej możliwości zmiany pojedynczego parametru zapytania).

Żadna z wyżej wymienionych aplikacji nie umożliwia prostego wykonywania zapytań do bazy, wygodnego sposobu zachowywania wcześniej wykonanych zapytań, bądź zapytań różniących się jedynie wartościami parametrów.

Rozdział 5

Prezentowane rozwiązanie

Użytkownik wymaga prostego interfejsu, którego obsługi nie musiałby się uczyć. Najprostszym i najpowszechniejszym formalnym sposobem przekazywania informacji są formularze. Za ich pomocą można automatycznie przetwarzać podane przez użytkownika informacje. Formularze w moim rozwiązaniu pomagają określić użytkownikowi jakiego rodzaju dane go interesują. Ważne jest, by użytkownik mógł za pomocą aplikacji zadać wszystkie istotne i sensowne logicznie zapytania, a jednocześnie nie zagubił się w nadmiernie wielu, początkowo nieprzydatnych opcjach. Jak dowodzi przykład opisanych powyżej aplikacji (zob. rozdz. 4) rozwiązanie, które by umożliwiało zadanie każdego zapytania (także tych "bez sensu") pozwala użytkownikowi na wykonanie wielu różnych akcji, podczas gdy zbiór tych naprawdę istotnych jest o wiele mniejszy. Biorąc pod uwagę ten fakt należałoby się zastanowić jakie istotne zapytania system powinien udostępniać użytkownikowi. Takie działania podejmuje większość zespołów programistycznych przygotowujących interfejs dla użytkowników. Minusem tego podejścia jest to, że kiedy w bazie danych zachodzą zmiany wymaga to zmian w kodzie aplikacji dostępu do bazy danych. Problem staje się poważniejszy, gdy projekt systemu ciągle się rozwija.

Trzeba zatem zminimalizować liczbę zmian w aplikacji, których by wymagała zmiana w bazie danych. W tym celu stworzę język opisu formularzy zapytania oraz system umiejący interpretować ten język, wyświetlać na jego podstawie formularze a następnie je przetwarzać. Głównym wymaganiem dla języka formularzy jest:

- wykonywanie każdego zapytania,
- umożliwianie parametryzowania zapytania,
- ma być na tyle prosty, by nie sprawiało kłopotu napisanie nowego formularza osobie potrafiącej układać zapytania SQL.

Format wszystkich dokumentów XML opisany jest przy użyciu standardu XML Schema ([XMLS]).

W prezentowanym rozwiązaniu można wyróżnić trzy podstawowe elementy: język interakcji z bazą danych, logikę i interfejs. Poniższe rozdziały zawierają ich szczegółowy opis.

5.1. Język interakcji z bazą danych

Do opisu języka posłużę się meta-językiem XML (zob. [XML]) . Ma on zdecydowane plusy w stosunku do np. opisu tekstowego ze względu na:

- czytelność zapisu informacji dla oka człowieka i dostępne edytory,

- możliwość jednoznacznego określenia formatów dokumentów,
- ogólną popularność,
- wiele istniejących technologii wspierających korzystanie z XML'a.

5.1.1. Formularze

Każdy formularz znajduje się w osobnym pliku i jest jednoznacznie identyfikowany poprzez jego nazwę. Formularze są podzielone na grupy i uporządkowane drzewiasto w celu łatwiejszej nawigacji.

Rozwiązanie zawarte w pracy ogranicza się do operacji wyszukiwania danych, jednak język ten można rozbudować do wykonywania także innych operacji na bazie danych.

Podstawowy formularz

Celem pracy jest to, żeby aplikacja umożliwiała wykonywanie wszystkich interesujących zapytań. Formularz zawiera opis zapytania. Użytkownik podaje jedynie parametry zapytania. Elementy, które można sparametryzować to konkretne wartości, które będą podawane przez użytkownika tj. napisy, liczby, daty. Użytkownik może też decydować jakie pola zobaczy w wyniku. W tym celu formularz będzie miał wyróżnione pola wejściowe, odpowiadające parametrom zapytania oraz pola wyjściowe odpowiadające nazwom kolumn w wyniku. Zapytanie jest pisane w prawie niezmienionej postaci, jednak zamiast konkretnych wartości są wstawiane wartości pól, a po instrukcji select nie ma listy kolumn, tylko zmienna oznaczająca, że w to miejsce ma być wstawiona lista (np. select $output from Tabela where nazwa='$nazwa'). Elementy wyświetlane, czyli pola, muszą też mieć opis, który byłby zrozumiały dla użytkownika. Przykładowy formularz można zobaczyć w pliku dołączonym do pracy (o nazwie „najjaśniej świecące.xml").

Rozwiązanie to ma jednak wady: wszystkie pola formularza muszą być wypełnione przez użytkownika, nie może on zrezygnować z określania np. wieku gwiazdy. Użytkownik nie ma też wpływu na takie elementy zapytania jak sortowanie wyniku, czy ustalenie bardziej złożonych zależności.

Formularz z warunkami

Chcąc zapobiec wpisywaniu przez użytkownika wszystkich wartości formularza, można zdefiniować dodatkowe zmienne, które określałyby, które warunki użytkownik chce brać pod uwagę, a które nie przy wykonywaniu zapytania. Zmienne takie mogłyby mieć przypisany kod, który powinien być użyty dla obu przypadków, np:

```
<condition name="name1" logicOperator="and"
 ifTrue="name='$nazwaGwiazdy'">
   <inField name="nazwaGwiazdy" .../>
</condition>
```

Ten fragment odpowiadałby sytuacji: jeśli użytkownik wybrał wyszukiwanie po tym warunku, to do zapytania w części "where" należy dodać warunek "and name = 'NazwaGwiazdy' ".

Minusami tego rozwiązania jest dodatkowa komplikacja formularza. W przypadku bardziej złożonych zapytań treść warunków mogłaby się stać zawiła.

Parsowanie zapytania

Tworzenie drzewa składni zapytania mogłoby wykluczyć problem wpisywania wartości wszystkich parametrów - w przypadku pominięcia jakiejś wartości parser odcinałby gałęzie, które je zawierają. Aby uniknąć wykonywania zapytań bez sensu wystarczyłoby określać, które pola są konieczne do wypełnienia.

Wadą tego rozwiązania jest to, że języki zapytań baz danych różnią się od standardu, co wymusza tworzenie dla każdej bazy danych osobnego parsera.

Podsumowanie

Zależnie od rodzaju zapytania oraz stopnia jego komplikacji różne rozwiązania mogą być bardziej lub mniej przydatne. Dla prostszych zapytań rozwiązanie pierwsze jest zupełnie wystarczające, dla tych najbardziej skomplikowanych konieczne będzie rozwiązanie trzecie.

Stworzona przeze mnie aplikacja obsługuje pierwszy rodzaj zapytania, jednak można ją rozszerzyć o inne rodzaje formularzy.

5.1.2. Wypełnienia

W plikach XML są przechowywane także wypełnienia formularzy. Dzięki dodatkowym formatom dla wypełnień formularzy nie są powielane podstawowe informacje o formularzu, ani opisy pól. Wypełnienia zawierają tylko nazwy pól i odpowiadające im wartości.

5.1.3. Połączenia

Pliki połączeń, także reprezentowane przez pliki XML'owe, opisują:

- typ połączenia, odpowiadający któremuś z istniejących rodzajów połączeń (każdy rodzaj połączenia jest pluginem),
- parametry danego połączenia (np. „user", „login").

Format opisu połączenia jest bardzo ogólny, dzięki temu może zawrzeć różne parametry, także dla połączeń nietypowych - komunikujących się z innym serwerem niż bazodanowy. Format połączenia opisany jest w pliku connectionFormat.xsd.

5.2. Logika

5.2.1. Połączenie z bazą danych

Najważniejszym zadaniem logiki jest wykonywanie zapytań. Dzięki w pełni konfigurowalnym połączeniom możliwe jest łączenie się z różnymi bazami danych a także z różnymi rodzajami baz danych. Parametry połączeń są wczytywane podczas startu aplikacji. Lista połączeń jest zapamiętywana i zwracana gdy użytkownik chce wybrać któreś dla danego zapytania.

Dzięki oddzieleniu definicji zapytania i połączenia, użytkownik może zadawać to samo zapytanie do różnych baz danych bez konieczności definiowania go na nowo. Ważnym elementem jest łatwe dodawanie nowego połączenia lub zmiana parametrów już istniejących połączeń. Ich parametry są przechowywane w plikach XML'owych o prostym formacie (zob. 5.1.3).

Program ma wbudowaną obsługę bezpośredniego łączenia się z najpopularniejszymi bazami danych (m.in. MySQL, PostgreSQL). Możliwe jest jednak dodanie własnej klasy obsługującej inny typ połączenia. Jeśli baza danych wymaga połączenia np. niebezpośredniego można dołączyć własne rozwiązanie.

5.2.2. Lista zadań

Kolejną cechą programu jest wielowątkowość dostępu do baz danych. Użytkownik oczekując na wynik zapytania, który składa się z wielu rekordów, może w tym samym czasie pracować na innych danych, wykonując inne zapytanie z tej samej lub innej bazy danych. Jest to szczególnie wygodne zwłaszcza przy pracy z dużymi bazami danych.

Po zleceniu wyszukiwania dodawane jest nowe zadanie do listy zadań, oraz tworzony nowy wątek połączenia. Na liście znajdują się zapytania na różnych etapach wykonywania. Po zakończeniu pobierania wyniku z bazy wątek zmienia stan zadania na zakończony. Dzięki temu użytkownik wie na jakim etapie jest zlecone zapytanie, a po jego zakończeniu od razu jest o tym powiadamiany i może pracować na dostarczonych danych. Możliwe oczekiwane wyniki to sukces i porażka, jeśli wystąpił błąd w połączeniu lub podczas wykonywania zapytania.

5.2.3. Opcje połączenia

Ze względu na to, że aplikacja może służyć do pracy z dużymi bazami danych wprowadziłam ograniczenia na maksymalny czas oczekiwania na wynik z bazy, oraz na maksymalną ilość wierszy wyniku. Oczywiście, jeśli użytkownik liczy się z długim czasem przesyłu danych i zależy mu na nich, może czekać na wynik do skutku.

5.2.4. Parametry zapytań

Program wspiera podstawowe typy danych (tzn. int, double, date, string, boolean). Jeśli istnieje potrzeba użycia innych typów danych trzeba dostarczyć odpowiednią klasę (plugin).

5.2.5. Wypełnienia formularzy

Zapytania dotyczące rekordów z wieloma polami mogą wymagać podania wielu parametrów zapytania. Gdy użytkownika interesują dane z jednego zakresu może wydawać mu się żmudnym wypełnianie na nowo za każdym razem pól formularza. Dodatkowym ułatwieniem dla użytkownika w takim przypadku jest możliwość zapamiętywania formularzy z wypełnionymi polami. Jeśli użytkownik chce w przyszłości powtórzyć dane zapytanie, możne je zapisać jako wypełniony formularz. Przy ponownym uruchomieniu programu ma możliwość wybrania formularza wraz z wypełnieniem pól.

5.2.6. Zapisywanie wyniku

Wynik zapytania można zapisać do pliku np. w celu dalszej analizy poprzez zaawansowane programy analityczne. Format pliku można wybrać z listy. Dodanie nowego formatu zapisu nie jest skomplikowane. Wystarczy dostarczyć klasę, która implementuje daną metodę zapisu.

5.3. Interfejs

Opisywany w pracy interfejs jest przykładowym interfejsem graficznym dla opisanej logiki, umożliwiający pełne wykorzystanie jej funkcjonalności.

Interfejs wykonałam w formie aplikacji, odrzucając opcję implementacji interfejsu jako serwis WWW. Decyzja taka podyktowana została zbyt dużą złożonością interfejsu (w rezultacie wizualizacja HTML'owa okazałaby się mało praktyczna). Wygoda korzystania była jednym z priorytetów przy projektowaniu systemu. Wiele opcji (zaimplementowanych i planowanych) korzysta z zasobów lokalnych klienta, co tym bardziej skłania w kierunku aplikacji, a nie interfejsu WWW. Ponadto aplikacja jest łatwiejsza przy wdrażaniu, nie trzeba uruchamiać serwera WWW lokalnie, ewentualnie wymagać połączenia sieciowego z serwerem.

5.3.1. Główne okno

Główne okno aplikacji otwiera się na starcie systemu. Jego podstawowym zadaniem jest wyświetlanie formularza w przejrzysty sposób, a po wybraniu opcji wyszukiwania sprawne operowanie na wynikach, w tym ich wyświetlanie. Okno główne składa się z menu, z którego jest dostępna większość opcji oraz z paska zakładek, do którego są dołączane formularze oraz wyniki. Dzięki temu użytkownik może operować na kilku z nich jednocześnie.

5.3.2. Struktura formularzy

Zbiór formularzy jest wyświetlany w postaci drzewa. Dzięki pogrupowaniu tematycznie formularzy, łatwiejsza jest nawigacja. Każdy z nich ma listę swoich wypełnień, jeśli zostały określone przez użytkownika. Zatem przy otwieraniu można wybrać pusty lub wcześniej wypełniony przez siebie formularz. Można też oczywiście wybrać wypełnienie po otwarciu pustego formularza.

Formularz jest wyświetlany jako zakładka w oknie głównym. Jej układ odpowiada polom formularza z pliku XML. Ma zatem listę checkbox'ów (komponentów) oznaczających pola wyjściowe oraz pola określające parametry zapytania. Każdy formularz ma dodatkowo listę połączeń, z której można wybrać na której bazie zapytanie będzie zrealizowane.

5.3.3. Prezentacja wyników

Do paska zakładek okna głównego mogą być także dołączane wizualizacje wyników. Podstawowa forma ich wyświetlania jest zrealizowana właśnie jako zakładka. Istnieje jednak możliwość dodania nowych form wizualizacji, zdefiniowanych przez użytkownika. Konkretny sposób prezentacji wyników jest reprezentowany przez plugin i jest obsługiwany przez system pluginów. Nowy rodzaj prezentacji wyników może być kolejnym rodzajem zakładki, lecz nie musi (może być np. otwierany w nowym oknie).

5.3.4. Lista zadań

Oprócz okna głównego z zakładkami formularzy oraz ich wyników, ważnym elementem jest lista zadań. Z niej użytkownik może się dowiedzieć na jakim etapie wykonywania jest zapytanie (zob. 5.2.2). Udostępnia ona także podstawowe operacje na wynikach tj. zapis do pliku, wizualizację oraz usunięcie danych.

5.4. Elementy rozszerzalne

W wielu miejscach program dostarcza jedynie podstawową funkcjonalność, jednak umożliwia dodawanie własnego kodu, który by ją rozszerzał. Aby ułatwić dostosowywanie aplikacji do potrzeb związanych z konkretnym zastosowaniem, dodałam element odpowiedzialny za

to, jakim jest system pluginów. Dzięki niemu wystarczy w odpowiednim katalogu umieścić skompilowane klasy implementujące określny typ pluginu. Klasy zostaną automatycznie załadowane przy starcie programu, a co za tym idzie użytkownik będzie mógł korzystać z nowo-dodanej funkcjonalności. Klasy ładowane są dynamicznie przez witrualną maszynę Javy (zob. [DYN]). Bardziej rozbudowany, choć działający na bardzo podobnej zasadzie system pluginów to np. *Java Plugin Framework* (zob. [JPF]).

Aplikacja na starcie sprawdza i wczytuje wszystkie zewnętrzne klasy, które są zgodne z określonymi interfejsami i znajdują się w przeznaczonych dla nich katalogach. Automatycznie dołącza dodatkowe opcje do list wyboru danej funkcjonalności.

Elementami, które można dodawać za pomocą systemu pluginów są:

- formaty zapisu wyszukanych w bazie wyników,
- sposoby łączenia z bazą (wbudowaną metodą połączenia jest bezpośrednia komunikacja),
- możliwe wizualizacje wyników,
- obsługa różnych typów parametrów (np.: date, int),
- pola odpowiadające parametrom zapytania w interfejsie (odpowiadają typom z punktu powyżej).

5.5. Implementacja

Aplikacja została napisana w języku Java 1.4. Głównym powodem jego wyboru była potrzeba, aby system był przenośny między systemami operacyjnymi.

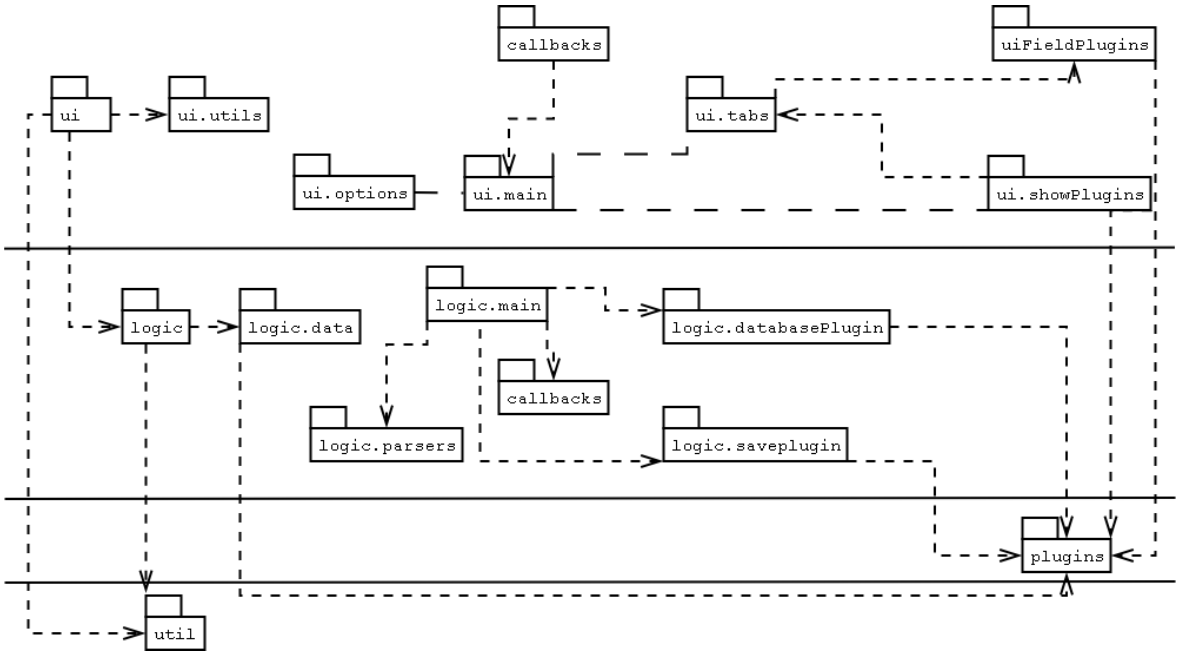
Pisząc program zwracałam szczególną uwagę na jego obiektowość. Podejście obiektowe uwidacznia się nie tylko w wyborze obiektowego języka, ale także w zastosowaniu wzorców projektowych (zob. [COO]). Oto lista najważniejszych wzorców projektowych zastosowanych w projekcie wraz z krótkimi opisami:

- klasa logic jest **fasadą** logiki; reprezentuje całą logikę systemu zamykając jej interfejs w zbiór kilku metod (upraszczając przy tym znacząco korzystanie z logiki),
- klasa Task jest klasą obserwowaną przez ResultPanel, która to implementuje interfejs callback'u (wzorzec **obserwatora**),
- **metoda produkcyjna** to wzorzec, który pomógł w zaprojektowaniu powiązań pluginów o konkretnej funkcjonalności i klas, za które są odpowiedzialne, np. twórca DatabasePlugin tworzy różne rodzaje produktu typu Database,
- pola UiField adoptują klasy biblioteki Swing do wyświetlania danego typu pola (wzorzec **adaptera**),
- MainWindow jest jedynym obiektem głównego okna aplikacji - a więc tzw. **singletonem**.

Zastosowanie wzorców projektowych w wielu wypadkach uprościło kod i sprawiło, że jest znacznie łatwiej rozszerzalny.

5.5.1. Podział na pakiety

W programie można wyróżnić cztery główne pakiety: logic, ui, plugins i utils. Poniżej omówię zadania każdego z nich. Diagram 5.5.1 przedstawia zależności pomiędzy nimi.

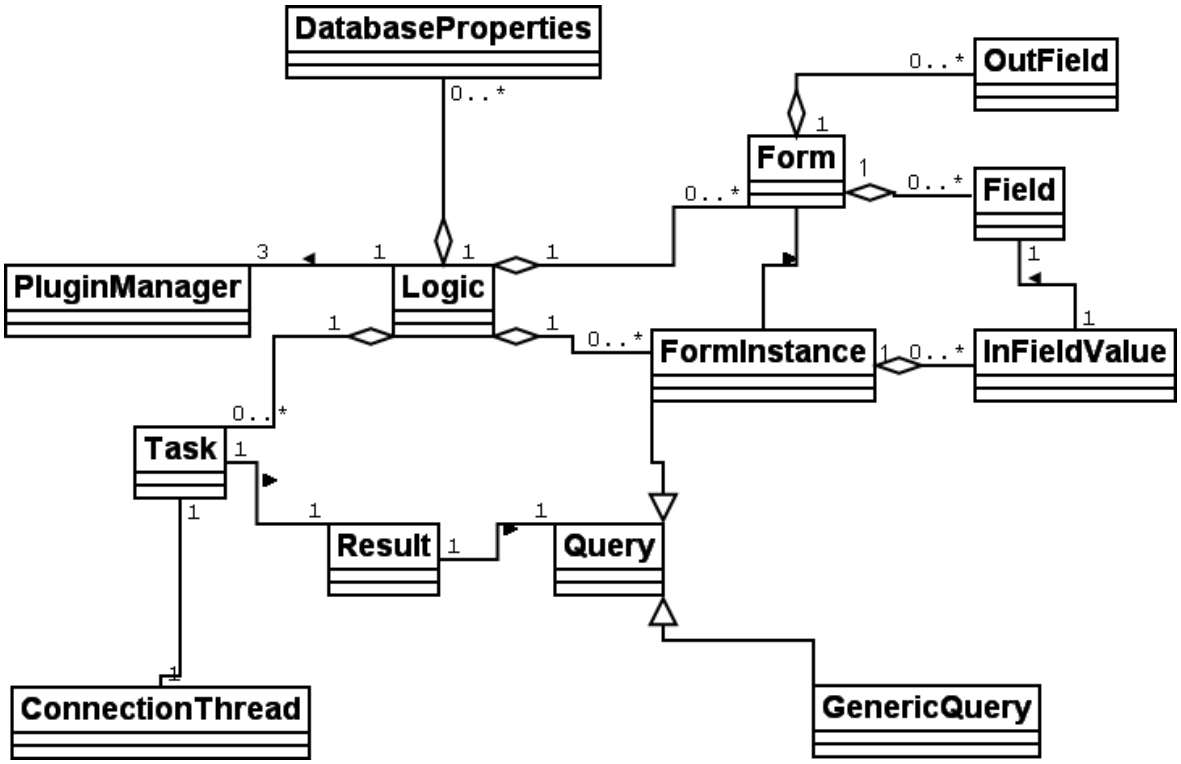


Rysunek 5.1: Uproszczony diagram pakietów.
Grubsze linie bez strzałek oznaczają powiązania w obie strony. Poziome linie oznaczają podziały podpakietów na 4 główne pakiety. Zależności między podpakietami i pakietami zaznaczyłam tylko jeśli wszystkie inne podpakiety korzystają z określonego podpakietu.

logic

Pakiet ten implementuje całą funkcjonalność logiki omówione w rozdziale 5.2. Zawiera podpakiety:

- logic.main
  Zawiera najważniejsze klasy logiki, w tym m.in.:
  - Logic, stanowiącą fasadę całej logiki systemu,
  - ConnectionThread implementującą łączenie się i wykonywanie zapytań do bazy danych w osobnym wątku,
  - Task, której obiekty reprezentują pojedyncze zadanie do wykonania na bazie danych (zob. 5.2.2).
- logic.callbacks
  W pakiecie tym znajdują się interfejsy spełniające funkcje callbacku, gdy logika potrzebuje dodatkowych informacji od użytkownika (np. przy niepełnej informacji o połączeniu, może żądać określenia parametrów np. użytkownika czy hasła).



Rysunek 5.2: Diagram klas ukazujący najważniejsze zależności pomiędzy klasami logiki.

- logic.data
  W tym pakiecie znajdują się proste klasy przechowujące jedynie dane. Dzielą się na następujące podpakiety:
  - logic.data.fields, zawiera klasy pól formularza:
    - OutField, odpowiada polu wyjściowemu formularza,
    - InFieldValue, odpowiada polu wejściowemu dla instancji formularza.
    - pakiet logic.data.fields.fieldPlugin, zawierający plugin i generowene przez niego pola wejściowe formularza (oraz wbudowane pluginy generujące typy: int, real, text, date; jego funkcje w systemie opisane zostały w rozdziale „Parametry zapytań", zob. 5.2.4),
  - logic.data.query, zawiera klasy:
    - abstrakcyjną Query - przechowuje parametry zapytania,
    - GenericQuery (dziedziczącą po Query) - przechowuje zapytanie SQL, które bezpośrednio zostało napisane przez użytkownika (w formie tekstowej),
    - Form - przechowuje wszystkie parametry formularza (odpowiada formularzowi z pliku XML),
    - FormInstance (dziedziczącą po Query) - odpowiada wypełnieniu formularza, zawiera wartości parametrów (zob. 5.2.5),
  - klasę Result, przechowującą wynik zapytania wraz z parametrami,
  - klasę DatabaseProperties, która przechowuje wszystkie parametry jednego połączenia (jest odpowiednikiem pliku XML opisującego połączenia; zadanie jakie pełni ona w systemie opisane jest w rozdziale „Połączenie", zob. 5.2.1),

- logic.parsers
  Zawiera klasy parsujące pliki XML formularzy i połączeń, umiejące interpretować znaczenia elementów i tworzące z nich obiekty (np. obiekty klasy Form) - odpowiedniki plików XML.

- logic.databasePlugin
  Zawiera abstrakcyjną klasę pluginu, który określa interfejs dla klas obsługujących różne rodzaje połączeń oraz przykład wbudowanego połączenia (które bezpośrednio wykonuje zapytanie w bazie danych). Szczegółowy opis tego, jaką funkcję pełni opisywany pakiet w systemie, znajduje się w rozdziale „Połączenie" (zob. 5.2.1).

- logic.savePlugin
  Zawiera abstrakcyjną klasę pluginu, zapisującą wynik zapytania. Pakiet ten ma także klasy implementujące zapis do pliku w najprostszej postaci tekstowej (wiersze rozwiązania są zapisywane w kolejnych liniach, kolumny przedzielone są znakiem';') oraz w formacie HTML. (zob. 5.2.6).

ui

Pakiet ten implementuje przykładowy interfejs umożliwiający wykonywanie zadań logiki. Dzieli się na następujące pakiety:

- ui.main
  Zawiera główne klasy interfejsu, czyli:

  - MainWindow – klasa odpowiadająca za główne okno aplikacji, implementuje też podstawowe akcje przez siebie udostępniane. Składa się z menu oraz części z zakładkami (zob 5.3.1).
  - ResultPanel – klasa odpowiadająca części okna wyświetlającej zlecone zadania na różnych etapach wykonania (w trakcie, sukces lub porażka, zob. 5.3.4).

- ui.tabs
  Zawiera abstrakcyjną klasę Tab, po której muszą dziedziczyć zakładki dołączane do panelu zakładek w oknie głównym (m.in. implementować menu „Tab" okna głównego i realizować funkcjonalność w nim dostępną). Ma także klasy, których obiekty wyświetlają formularz (zob. 5.3.2), zakładkę z polem do wpisania przez użytkownika zapytania SQL, oraz zakładkę wyświetlającą wynik zapytania w tabeli.

- ui.options
  Pakiet ten zawiera klasy pomocnicze, które realizują część opcji dostępnych w menu okna, są to w większości klasy wyświetlające dodatkowe okienka, m.in.:

  - ConnectionOptionWindow, klasę okienka w którym można ustawić maksymalny czas i wielkość wyniku,
  - SelectResultsTypeWindow, odpowiadającą okienku, w którym można wybrać format zapisu pliku lub rodzaj wizualizacji,
  - Pakiet ui.options.open, w którym są klasy umożliwiające otwarcie formularza i wypełnienia jego pól (zob. 5.3.2)
  - inne klasy pomocnicze, umożliwiające określenie użytkownikowi parametrów wykonania zapytania.

- ui.callbacks
  Zawiera klasy implementujące interfejsy callbacków logiki, umożliwiające logice pobranie od użytkownika dodatkowych wartości, wyświetlają dodatkowe okienka.

- ui.uiFieldPlugin
  Zawiera abstrakcyjną klasę pluginu generującego pola interfejsu operujące na wartościach formularza, oraz wbudowane pola odpowiadające polom formularza z logiki (zob. opis pakietu logic.data.fields.fieldPlugins w rozdz. 5.5.1).

- ui.showPlugin
  Zawiera abstrakcyjna klasę metody wizualizacji wyniku oraz wbudowaną implementację wizualizacji wyniku w postaci HTML (zob. 5.3.3).

- ui.utils
  Zawiera klasę wyświetlającą informację o niezgodnościach w systemie, także o błędach krytycznych.

plugins

Implementuje system pluginów. Umożliwia dynamiczne pobranie klas implementujących plugin z odpowiednich katalogów na dysku.

utils

Zawiera klasy pomocnicze wykorzystywane w systemie, np. konwertery typów.

5.5.2. Przykładowa akcja systemu – wyszukiwanie

Następujące operacje wstępne musza być wykonane, aby wyszukać dane w bazie:

- aplikacja musi być włączona,

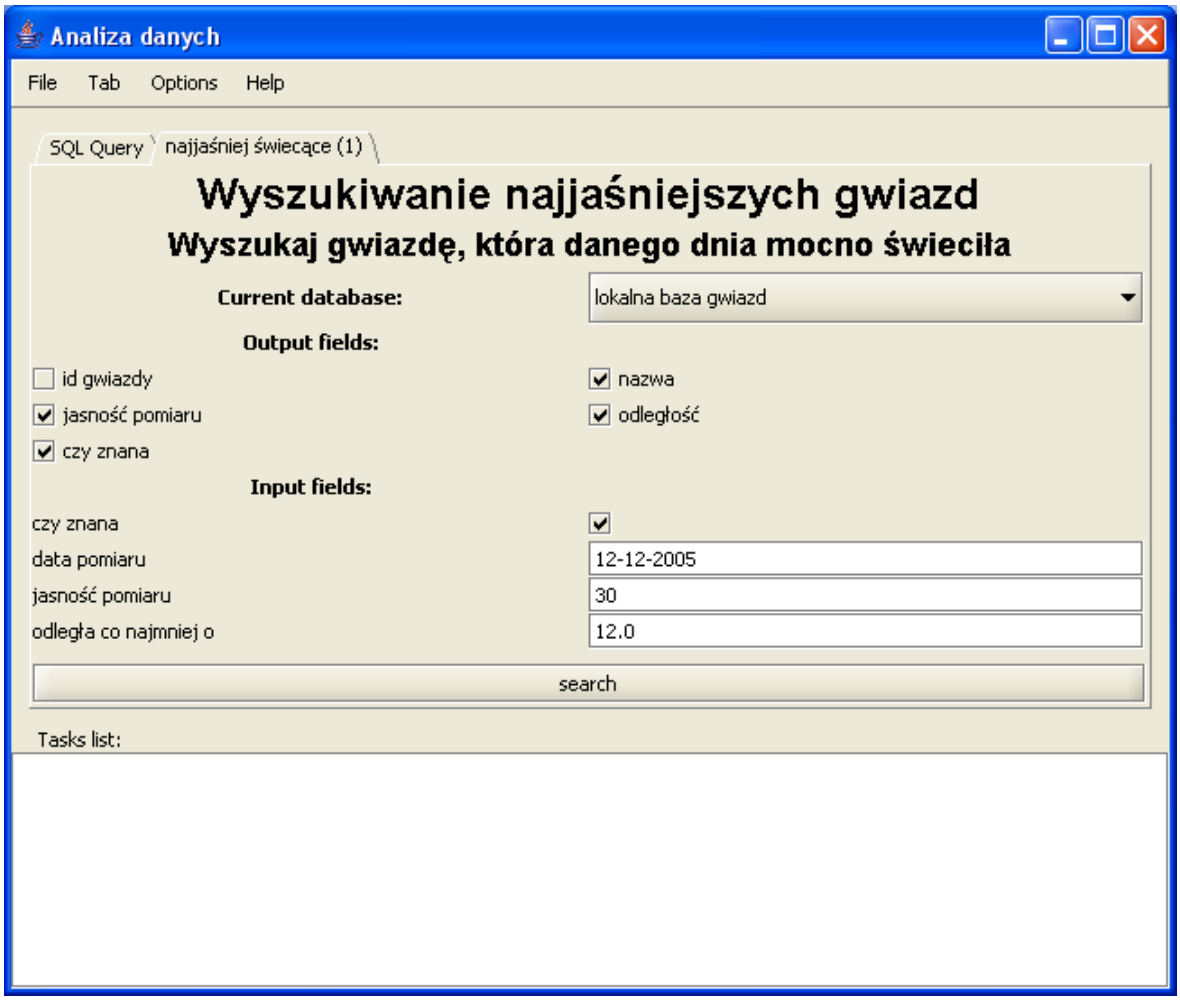
- otworzony jest formularz lub zakładka bezpośredniego zapytania SQL.

Po wypełnieniu pól (w przypadku formularza) lub wpisaniu ręcznie zapytania SQL'owego (w przypadku zakładki bezpośredniego zapytania SQL) i przyciśnięciu przycisku „search" program rozpoczyna akcję wyszukiwania w bazie danych.

Interfejs, tj. moduł ui projektu, korzysta z logiki. Logika pobiera od PluginManagera odpowiedzialnego za połączenia z bazą danych plugin o wybranym typie (typ został określony w pliku XML połączenia, a następnie wybrany przez użytkownika z listy możliwych połączeń). Następnie tworzone jest nowe zadanie (obiekt klasy Task), które będzie odpowiedzialne za wyszukiwanie danych, po czym dołączane jest ono do listy zadań. Tworzony jest także specjalny wątek przeznaczony do wykonania zapytania. Połączenie z bazą danych lub inną aplikacją i oczekiwanie na wynik odbywa się w tymże wątku. Gdy rezultat zapytania zostanie dostarczony lub wystąpi błąd połączenia lub jakikolwiek inny błąd bazy danych, stan zadania ustawiany jest, odpowiednio, na sukces lub porażkę. Zadanie, gdy zmienia swój stan informuje o tym interfejs (callback), dzięki czemu użytkownik na bieżąco śledzić może stan wykonywanych zapytań.

Powiadomiony o zmianach w zadaniu interfejs odświeża stan zadania. Od tej chwili użytkownik może oglądać lub zapisywać wynik na dysku.

Akcję wyszukiwania można zobaczyć na diagramie sekwencji 5.5.2.



Rysunek 5.3: Aplikacja gotowa do wykonania akcji wyszukiwania.

5.5.3. Użyte technologie

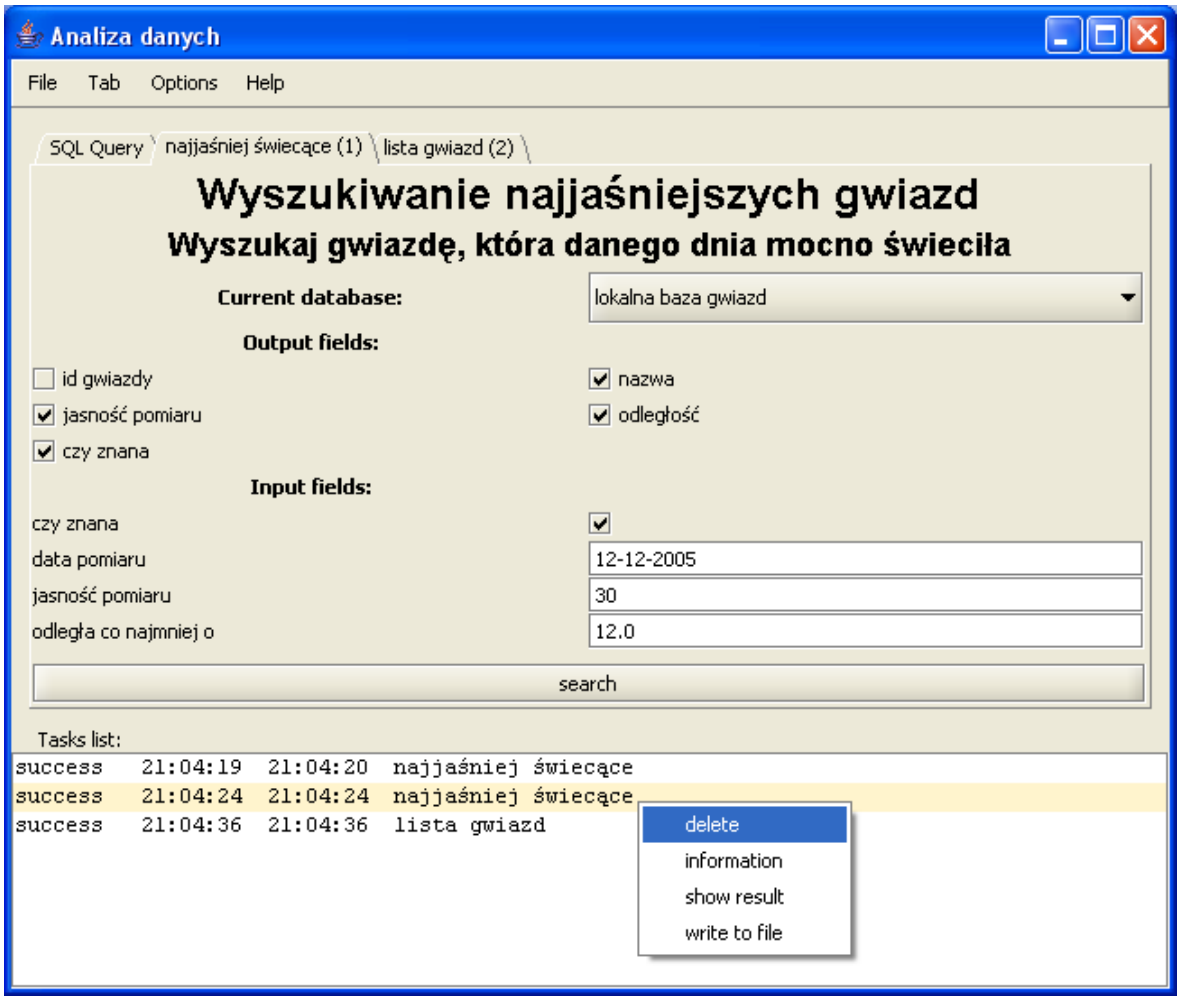
Do komunikacji z bazą danych aplikacja wykorzystuje technologię JDBC (zob. [JDB]), zapewniającą jednolity dostęp do różnych systemów bazodanowych. Różne systemy baz danych (np. Oracle, Postgres) wymagają jedynie zmiany sterownika JDBC. Ponadto jest najbardziej wydajna i najszerzej wspierana spośród istniejących technologii.

Do parsowania dokumentów XML korzystam z parsera DOM (zob. [ELL]), który umożliwia wieloetapowe przetwarzanie dokumentów XML.

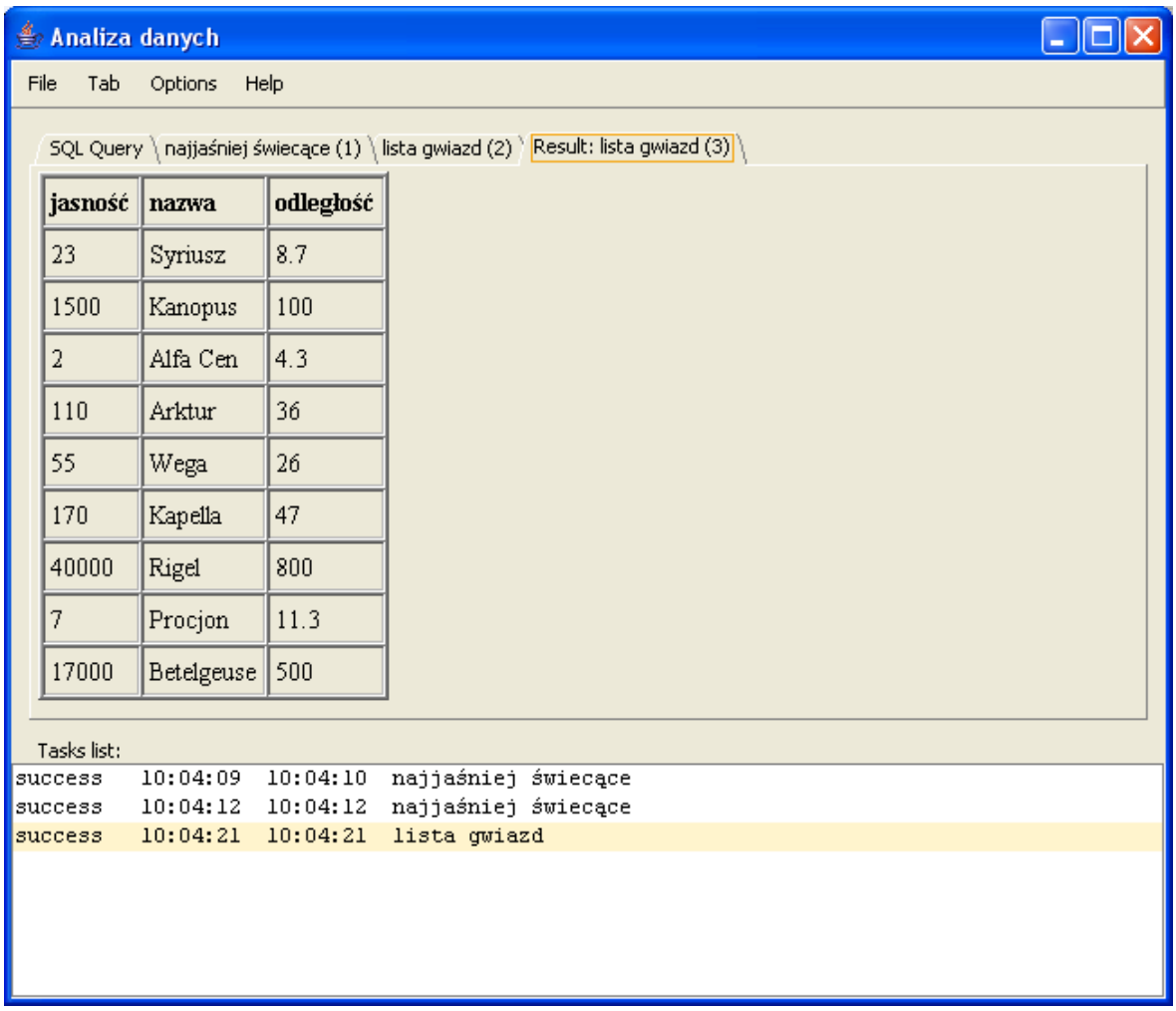
Interfejs korzysta z biblioteki Swing (zob. [SWI]) oraz JGoodies (zob. [JGO]).

5.5.4. Dokumentacja użytkownika

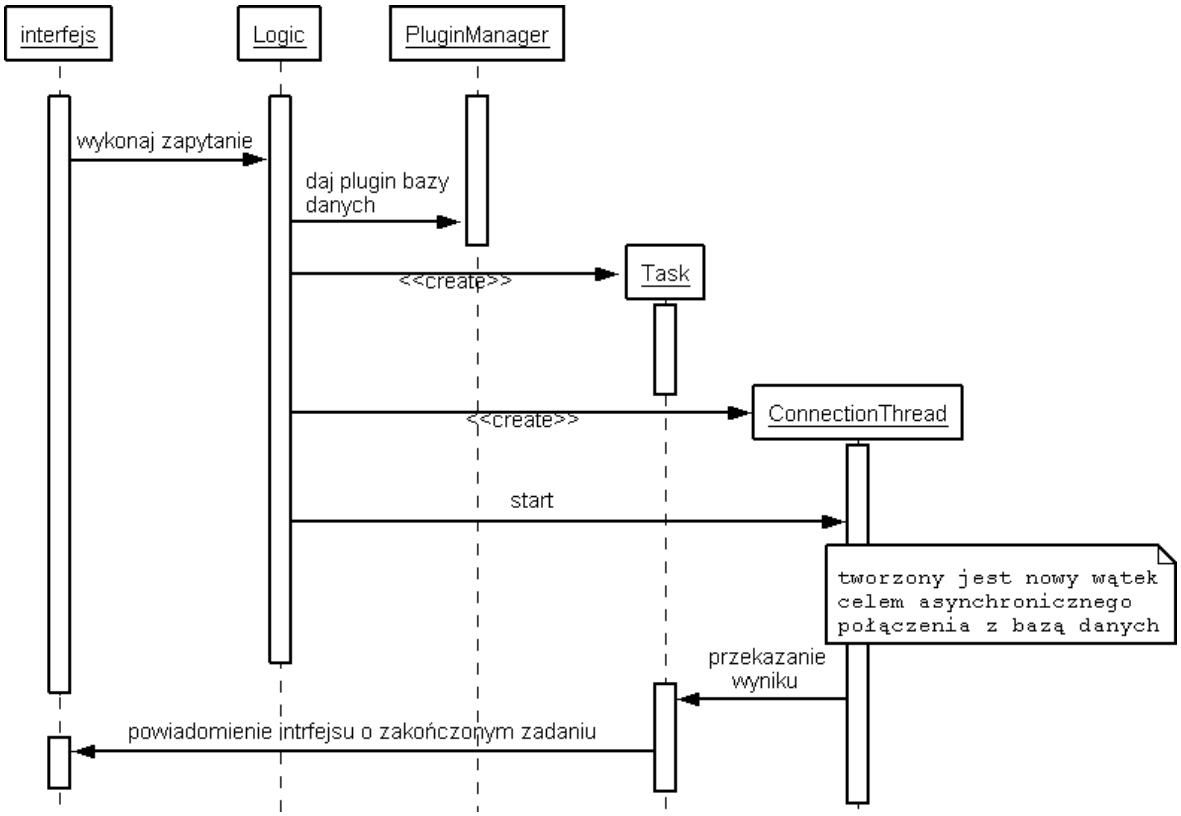
Do pracy załączam dokumentację użytkownika końcowego (wyszukującego dane w bazie) oraz dokumentację dla osoby nadzorującej wykorzystanie aplikacji (w tym tworzenie i zmianę formularzy, oraz dodawanie do aplikacji nowych właściwości, tzw. wtyczek).



Rysunek 5.4: Aplikacja po wykonaniu akcji wyszukiwania.



Rysunek 5.5: Aplikacja wyświetlająca wynik zapytania.



Rysunek 5.6: Diagram sekwencji opisujący akcje wyszukiwania.

Rozdział 6

Możliwe rozszerzenia

Opisany przeze mnie pomysł umożliwia rozszerzanie istniejącej funkcjonalności w wielu zakresach. Poniżej omówię najciekawsze rozszerzenia.

6.1. Operacje na formularzach

Definiując dodatkowy element formularza, który określa metadane można udostępnić opcję wyszukiwania odpowiednich formularzy. Opcja przeszukiwania oczywiście mogłaby się opierać także na innych elementach dokumentów, czemu szczególnie sprzyja struktura plików XML. Na formularzach, które wymagałyby zapisywania drzewa składni SQL'owego zapytania lub też poprzez parsowanie go można wykonywać dodatkowe operacje na zapytaniach, określać rodzaj podobieństwa, wykonywać ciekawe wyszukiwania formularzy, które: korzystają z określonych tabel, biorą pod uwagę konkretną ilość parametrów lub też używają konkretnych operacji SQL'owych (zob. [MET]).

6.2. Automatyczne zapamiętywanie wartości parametrów

W obecnej postaci aplikacja umożliwia zapisywanie parametrów zapytania, jednak przydatną funkcją mogłoby się okazać zapamiętywanie automatyczne kilku (może wszystkich) wybieranych parametrów, aby można byłoby powrócić do wcześniejszych ustawień. Aplikacja mogłaby też zapamiętywać i podpowiadać wpisywane wartości.

6.3. Zapisywanie informacji o wynikach

Funkcja ta pozwalałaby na obejrzenie historii zmian w wynikach zapytania wykonywanych przez formularz.

6.4. Porównywanie wyników zapytań

Opcja ta umożliwiałaby proste porównanie wyników zapytania. Po wybraniu przez użytkownika rezultatów zapytań aplikacja sprawdzałaby, czy wyniki są takie same, zwracałaby statystyki, ile krotek przybyło i ile się zmieniło. Funkcja byłaby szczególnie przydatna, gdy wyniki pochodziłyby z różnych baz danych.

6.5. Automatyczna aktualizacja formularzy

W celu aktualizacji zbioru formularzy dostępnych w projekcie, aplikacja mogłaby automatycznie pobierać je ze wskazanej lokalizacji: mogłaby to by być wybrana baza danych lub ustalony URL. Aplikacja mając możliwość łączenia się z internetem aktualizowałaby strukturę formularzy. W podobny sposób osoby z jednego projektu mogłyby dzielić się wypracowanymi przez siebie ciekawymi zapytaniami.

Rozdział 7

Zastosowanie w projekcie „Pi of the sky”

7.1. Projekt „Pi of the sky”

Projekt „Pi of the sky” ([PIO]) jest projektem badawczym monitorującym niebo w celu poszukiwania błysków pochodzących z gigantycznych eksplozji pozagalaktycznych oraz badania obiektów astrofizycznych o szybkiej zmienności (1 s - 1 rok). Zespół stanowią głównie studenci i doktoranci Instytutu Problemów Jądrowych, Centrum Fizyki Teoretycznej PAN, Wydziału Fizyki UW, Instytutu Systemów Elektronicznych PW, Wydziału Fizyki PW i Uniwersytetu kard. St. Wyszyńskiego.

Od lipca 2004 w Chile na terenie obserwatorium Las Campanas należącego do Carnegie Institution of Washington działa prototyp. Dwie kamery (2000 x 2000 pikseli, 5 - 10 s naświetlania) robią zdjęcia niebu, a następnie fotografie te są poddawane wieloetapowej analizie. Wyniki analiz trafiają do bazy danych (Postgres - zob. [POS]). Są w niej trzymane głównie - znane i nieznane - zaobserwowane obiekty astrofizyczne oraz ich pomiary jasności.

7.2. Problemy z dostępem do danych

Obecnie dostęp do zebranych danych istnieje w następujących formach:

- Interfejs WWW
  Jest ogólnodostępny, wygodny dla użytkownika, umożliwia oglądanie podstawowych parametrów pomiarów. Jest to jednak rozwiązanie raczej ukazujące czym zajmuje się projekt „Pi of the sky”, niż pozwalające na pełne wykorzystanie wyników badań. Jego główne ograniczenia to:
  - tylko kilka rodzajów zapytań,
  - dodanie kolejnej właściwości wymaga dużo nakładu pracy (zmiana kodu przez programistę),
  - połączenie z określoną bazą danych (istnieje potrzeba porównania wyników z danymi z innych baz),
  - wyniki niewygodne dla dalszej analizy.
- Skrypty lub bezpośrednie wykonywanie zapytań na bazie danych
  Poza oczywistą niewygodą w korzystaniu, do wad tego rozwiązania należy:

– pisane przez każdego użytkownika „samemu sobie”,
– nie każdy zna SQL, zatem nowe osoby dołączające do projektu muszą się go uczyć od podstaw,
– brak dostępu dla osób spoza projektu.

7.3. Wykorzystanie aplikacji

Wdrożenie aplikacji, która jest tematem tej pracy w projekcie „Pi of the sky” ma ułatwić pracę z danymi użytkownikom. Dzięki niej nowe osoby nie będą zmuszone uczyć się SQLa. Użytkownicy za pomocą wygodnego interfejsu będą mogli przeglądać dane.

Obecnie osoby spoza projektu nie mają dostępu do wszystkich astrofizycznych danych (jedynie do tych, które udostepnia interfejs WWW). Dzięki wdrożeniu mojej aplikacji osoby spoza projektu będą miały możliwość zadawania zapytań dotyczących wszystkich udostępnionych danych.

W projekcie istnieje kilka baz danych roboczych. Dzięki liście połączeń w aplikacji użytkownicy mogą łatwo zmieniać bazę, na której ma być wykonane zapytanie.

7.4. Plany na przyszłość

Dane zbierane przez omawiany projekt mają być udostępnione (a przynajmniej ich część) dla użytkowników spoza projektu. Aby chronić dane, baza danych jest widoczna tylko z wewnętrznej sieci projektu. Konieczny jest pośrednik łączący aplikację działającą z zewnątrz z bazą danych. W tym celu postawiony jest serwer Tomcat. Najbliższe prace przy wdrożeniu polegają na napisaniu serwletu odbierającego zapytania z aplikacji, łączącego się z odpowiednią bazą oraz przesyłającego do aplikacji dane. Dostosowanie aplikacji do komunikowania się z serwerem oznacza dopisanie pluginu połączenia, który będzie klientem HTTP.

Rozdział 8

Podsumowanie

W pracy zostało przedstawione rozwiązanie ułatwiające wyszukiwanie danych. Zostało ono stworzone z myślą o projektach, w których wiele osób nie zna SQLa i dużą przeszkodą jest dla nich tworzenie własnych zapytań.

Jest to kompromis pomiędzy, z jednej strony, prostotą wykonywania wyszukiwań, a z drugiej możliwościami wyszukiwania. Interfejs, w którym końcowy użytkownik nie musi wiedzieć praktycznie nic o bazach danych był możliwy do stworzenia dzięki łatwemu komponowaniu nowych formularzy bez konieczności tworzenia dodatkowego kodu. Pomysł opiera się niejako na dwuetapowym procesie tworzenia zapytań: przy wdrażaniu - projektowanie przydatnych formularzy przez administratora (ale mogą one być dodawane także i później), oraz wypełnianie wybranymi parametrami przez użytkownika końcowego.

Dużą zaletą rozwiązania jest elastyczność w pracy z bazami danych. Opisywaną aplikację łatwo skonfigurować, by działała na wielu nawet różnych bazach danych.

Tworzenie własnego interfejsu dla projektu, wymaga dużo czasu i kosztów. Istniejące narzędzia są jednak skomplikowane dla przeciętnego użytkownika. Opisana aplikacja wypełnia lukę pomiędzy rozwiązaniami umożliwiającymi konstruowanie dowolnych zapytań (opisanymi w pracy), a interfejsami tworzonymi na potrzeby konkretnego projektu.

Znaczącym elementem pracy jest praktyczna implementacja opisanego pomysłu, która jest gotowym do użycia programem. Jej dodatkowym atutem jest ładne, obiektowe rozwiązanie, które umożliwia wszechstronną rozszerzalność.

W pracy zawarłam podstawowy pomysł na aplikację. Jak w przypadku każdego rozwiązania informatycznego, istnieje wiele możliwych ulepszeń czy dodatkowych opcji, które byłyby przydatne dla użytkownika. Te najciekawsze opisałam w rozdziale „Możliwe rozszerzenia".

Aplikacja ma swoją stronę WWW ([EAS]).

Dodatek A

connectionFormat.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
  <xs:element name="database">
    <xs:annotation>
      <xs:documentation>
       This document discribes the format of connection.
       The database element declares the root of the document.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
             The property may occur zero or more times.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>
           The type attribute indicates the type of the connection.
           This text string must be understood by the application.
           Known built-in type is DirectSQL type.
           This is required attribute.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="property">
    <xs:annotation>
      <xs:documentation>
       The property element declares one of the property of connection.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:attribute name="name" type="xs:ID" use="required">
```

```
        <xs:annotation>
          <xs:documentation>
           The name attribute is a text string containing
           the unique identifier of the property element.
           It must be unique within the instance document.
           This is required attribute.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="value" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>
           The value attribute is a value of (connection) property.
           This is required attribute.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Dodatek B

formFormat.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
  <xs:element name="form">
    <xs:annotation>
      <xs:documentation>
       This document describes the format of form.
       The form element declares the root of the document.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="output" minOccurs="1" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
             The output element contains output fields of form.
             It must occur exactly once.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element ref="input" minOccurs="1" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
             The input element contains input fields of form.
             It must occur exactly once.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element ref="query" minOccurs="1" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
             The query element declares SQL query, special query
             for this form. It must occur exactly once.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>
```

```
        The title element is the title of form.
        This is optional attribute.
       </xs:documentation>
     </xs:annotation>
   </xs:attribute>
   <xs:attribute name="comment" type="xs:string" use="optional">
     <xs:annotation>
       <xs:documentation>
        The comment attribute describes what kind of data
        the form can search. This is optional attribute.
       </xs:documentation>
     </xs:annotation>
   </xs:attribute>
 </xs:complexType>
</xs:element>
<xs:element name="output">
 <xs:annotation>
   <xs:documentation>
    The output element contains output fields of form.
   </xs:documentation>
 </xs:annotation>
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="outField" minOccurs="0" maxOccurs="unbounded">
       <xs:annotation>
         <xs:documentation>
          The outField element may occur any number of times.
         </xs:documentation>
       </xs:annotation>
     </xs:element>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="outField">
 <xs:annotation>
   <xs:documentation>
    The outField element describes one of the output
    fields of form.
   </xs:documentation>
 </xs:annotation>
 <xs:complexType>
   <xs:attribute name="name" type="xs:ID" use="required">
     <xs:annotation>
       <xs:documentation>
        The name attribute is a text string containing
        the unique identifier of the fields element.
        This value must be unique within the instance document.
        This is required attribute.
       </xs:documentation>
     </xs:annotation>
   </xs:attribute>
   <xs:attribute name="description" type="xs:string" use="required">
     <xs:annotation>
       <xs:documentation>
        The description attribute describes the meaning of
```

```
          the field for the end-user. This is required attribute.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="defaultValue" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>
         The defaultValue attribute contains default value
         for that field. This is optional attribute.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="input">
  <xs:annotation>
    <xs:documentation>
     The input element contains input fields of form.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="inField" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
           The outField element may occur zero or more times.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="inField">
  <xs:annotation>
    <xs:documentation>
     The inField element describes one of the input fields of form.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="name" type="xs:ID" use="required">
      <xs:annotation>
        <xs:documentation>
         The name attribute is a text string containing
         the unique identifier of the fields element.
         This value must be unique within the instance document.
         This is required attribute.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:token" use="required">
      <xs:annotation>
        <xs:documentation>
         The type attribute indicates the type of the value data.
         This text string must be understood by the application.
         Known built-in types are: text, date, int, real and boolean.
```

```
          This is required attribute.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="defaultValue" type="xs:string" use="optional">
       <xs:annotation>
         <xs:documentation>
          The defaultValue attribute contains default value
          for that field. This is optional attribute.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="description" type="xs:string" use="optional">
       <xs:annotation>
         <xs:documentation>
          The description attribute describes the meaning of the field
          for the end-user. This is required attribute.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>
   </xs:complexType>
 </xs:element>
 <xs:element name="query">
   <xs:annotation>
     <xs:documentation>
      The query element declares SQL query for this form.
     </xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:attribute name="text" type="xs:string" use="required">
       <xs:annotation>
         <xs:documentation>
          The text attribute contains parameterized SQL query
          (e.g. "select $output where name = $name and age = $age").
          Each parameter within the query (i.e. input field)
          is preceeded by the '$' character. After '$' follows
          the name of the parameter. The name of each parameter
          has to be the name of one of the input fields'.
          '$output' parameter within the query is an obligatory
          special parameter placed after 'select' and before 'where' -
          this is the place where all output fields should be
          automatically put when performing a query.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Dodatek C

najjaśniej świecące.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<form title="Wyszukiwanie najjaśniejszych gwiazd"
 comment="Wyszukaj gwiazdę, która danego dnia mocno świeciła"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="../formFormat.xsd">
  <output>
    <outField name="Gwiazdy.id" description="id gwiazdy"
     defaultValue="false"/>
    <outField name="nazwa" description="nazwa"
     defaultValue="true"/>
    <outField name="Pomiary.jasnosc" description="jasność pomiaru"
     defaultValue="true"/>
    <outField name="odleglosc" description="odległość"
     defaultValue="true"/>
    <outField name="znana" description="czy znana"
     defaultValue="true"/>
  </output>
  <input>
    <inField type="date" name="data" defaultValue="12-12-2005"
     description="data pomiaru"/>
    <inField type="real" name="odleglosc_wieksza" defaultValue="12"
     description="odległa co najmniej o"/>
    <inField type="int" name="jasnosc_pomiaru" defaultValue="30"
     description="jasność pomiaru"/>
    <inField type="boolean" name="czy_znana" defaultValue="30"
     description="czy znana"/>
  </input>
  <query text="select $output from Gwiazdy, Pomiary
   where odleglosc > $odleglosc_wieksza and
   znana = $czy_znana and Gwiazdy.id = Pomiary.id and
   Pomiary.data = '$data' and pomiary.jasnosc > $jasnosc_pomiaru
   order by pomiary.jasnosc"/>
</form>
```

Dodatek D

Zawartość płyty CD załączonej do pracy

src – katalog zawierający kod źródłowy aplikacji easySQL

bin – katalog zawierający aplikację w wersji wykonywalnej oraz instrukcję obsługi administratora i użytkownika

doc – katalog zawierający dokumentację kodu źródłowego

readme.txt – plik zawierający informacje odnośnie kompilacji kodu źródłowego i uruchamiania aplikacji

malgorzata_sawitus.pdf – elektroniczna wersja niniejszej pracy

Bibliografia

[COO] James W. Cooper, *Design Patterns*,
http://www.patterndepot.com/put/8/JavaPatterns.htm, 29.05.06

[DBA] DBACentral for MySQL,
http://www.myitforum.com/articles/14/view.asp?id=2239, 29.05.06

[DBX] DBxtra - Reporting Tool,
http://www.dbxtra.com/, 29.05.06

[DYN] Sheng Liang, Gilad Bracha Dynamic Class Loading in the Java Virtual Machine,
http://www.humbertocervantes.net/coursdea/DynamicClassLoadingInTheJavaVirtual-Machine.pdf, 29.05.06

[EAS] easySQL,
http://www.easySQL.end.pl, 29.05.06

[ELL] Elliotte Rusty Harold, *Processing XML with Java*,
http://www.cafeconleche.org/books/xmljava/, 29.05.06

[JDB] JDBC Technology,
http://java.sun.com/products/jdbc/, 29.05.06

[JGO] JGoodies,
http://www.jgoodies.com/index.html, 29.05.06

[JPF] Java Plug-in Framework,
http://jpf.sourceforge.net/api/index.html, 29.05.06

[MET] Jan Van den Bussche, Stijn Vansummeren, Gottfried Vossen, Meta-SQL: Towards Practical Meta-Querying,
Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings

[MIC] Microsoft Data Analyzer,
http://www.microsoft.com/office/dataanalyzer/default.asp, 29.05.06

[PIO] Pi of the sky,
http://grb.fuw.edu.pl/index.html, 29.05.06

[POS] PostgreSQL,
http://www.postgresql.org/, 29.05.06

[SWI] Swing,
http://java.sun.com/products/jfc/tsc/articles/architecture/, 29.05.06

[VIS] Visual SQL-Designer,
http://www.visualsoftru.com/sqldesigner.asp, 29.05.06

[XML] Extensible Markup Language,
http://www.w3.org/XML/, 29.05.06

[XMLS] XML Schema,
http://www.w3.org/XML/Schema, 29.05.06